Introduction to Java — PL/SQL Developers Take Heart!

Peter Koletzke

Technical Director &
Principal Instructor

QUOVERA

Dennis Should Know

A language that doesn't affect the way you think about programming is not worth knowing.
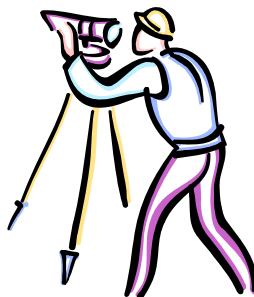
—Dennis M. Ritchie

QUOVERA

Survey

- Years with PL/SQL?
  - Less than 2, 2-4, 4-15
- Years with Java?
  - None
  - 2, 2-4, 4-12, 13+
- Other languages?
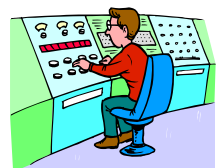 - C
  - C++
  - Smalltalk
  - COBOL, Basic, JCL, Perl …

QUOVERA

Agenda

- Java Basics

- Object Orientation

- Anatomy 101

- Datatypes and Variables

- Control Statements

Slides are available on the NYOUG and Quovera websites. White paper in the NYOUG Journal and on the Quovera website

QUOVERA

Java Advantages

- Platform independent (the promise of portability)
- The core of J2EE
  - Lots of deployment options
- Modern language
  - Object oriented (the promise of reuse)
  - Supports multi-tasking
- Looks like C++
  - No pointers
  - Manages memory for you (C-- ?)
- A well-developed user community
  - Open-source support
- Emerging language
  - Currently hot; It has Oracle's attention

Java Drawbacks

- Emerging language
  - Currently hot
  - No mass acceptance
    - Microsoft is still in the game (.NET languages)
  - Technologies are changing rapidly
- Not a "normal" skill for an Oracle developer
- It requires a 3GL tool
  - Some IDEs help create code
- Arguably more complex than PL/SQL
  - Database access is not a natural thing
 - Needs object-oriented thinking

Developing the Code

1. Create or modify source code file
   - Standard ASCII text – can use Notepad or vi
   - Use .java extension (HiThere.java)
2. Compile the source code file
   - `javac.exe HiThere.java`
   - Creates <classname>.class (HiThere.class)

> Called "bytecode" or "bytecodes"

3. Test the class file in a runtime (interpreter) session
   - Also called *Java Virtual Machine* (JVM)
   - `java.exe HiThere`
4. Repeat 1-3 until victory is declared
5. Deploy the file
   - Package with required libraries

Agenda

- Java Basics

- Object Orientation

- Anatomy 101

- Datatypes and Variables

- Control Statements
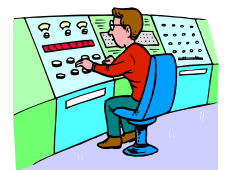
OO Basics

- Basic building block is the Class
  - The fundamental building block
  - A pattern, template, archetype, or blueprint from which objects are built
    - Like for a car – 1988 Honda Accord LXI
  - A "concept" not anything "real"
  - Also called an *abstract data type*
- Objects
  - "Real" things – in code, anyway
    - Like PL/SQL variables
  - The "instantiation" of a class
    - 1988 Honda Accord LXI (VIN 785789359019)
  - Kind of like a variable built from a data type

Big Three OO Concepts

- Inheritance
  - Parent-child relationship
  - Child has data and behavior of the parent
  - Classes inherit by "subclassing" a parent class
- Encapsulation
  - Data is hidden from the outside
  - Use an approved interface to get to data (setCity, getAddress, etc.)
- Polymorphism
  - Similar to overloading in PL/SQL
  - Caller doesn't know which method will be called
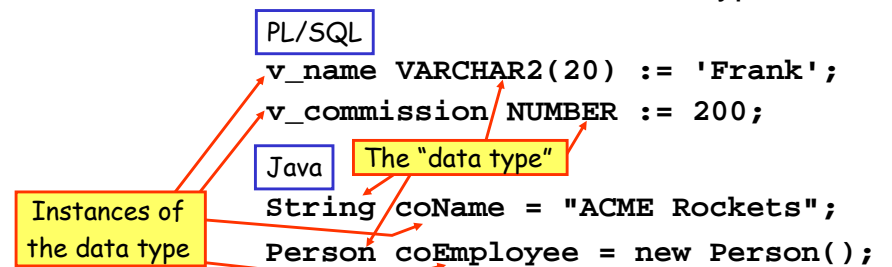
OO (Java) vs PL/SQL?

- PL/SQL does not have inheritance
  - Cannot subclass a procedure or package
  - You can call prebuilt code, of course
- OO objects vs PL/SQL variables
  - Behavior is loosely bound in PL/SQL
  - Behavior is integrated into OO objects
- Different paradigms

Another Way to Think About Objects

- It is like an abstract data type
  - Each "thing" created from the data type has the same characteristics as the data type

```
PL/SQL
v_name VARCHAR2(20) := 'Frank';
v_commission NUMBER := 200;
Java          The "data type"
String coName = "ACME Rockets";
Person coEmployee = new Person();
```

Instances of the data type

  - The difference is that Java (OO) has methods for the declared instance

Agenda

- Java Basics
- Object Orientation
- **Anatomy 101**
- Datatypes and Variables
- Control Statements
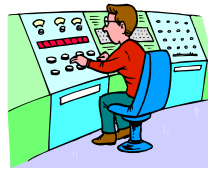
Flon's Law

There is not now, and
never will be, a language
in which it is
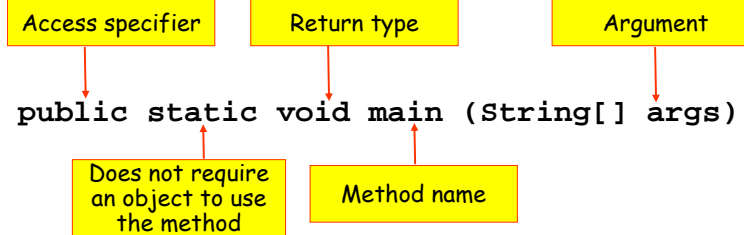the least bit difficult to
write bad programs.

Basic Java Terms

- Class
  - Fundamental building block
  - All code is contained in classes
  - Source code (.java) is compiled (.class)
- Object
  - An *instance* of a class
- Method
  - Unit of code contained in a class
  - Like PL/SQL procedures and functions
- Constructor
  - Code unit used to instantiate an object

About Methods

- Method signature:

Access specifier	Return type	Argument

```
public static void main (String[] args)
```

Does not require an object to use the method

Method name

- Access specifier declares which classes can see this class
  - E.g., "private" is not visible to other classes
- Return type can be something or nothing (void)
- Overloading allowed
  - More than one method with the same name and different arguments

About Constructors

- They look a bit like methods
- No return type (not even void)
  - For example, `Box(int quantity)`
- Responsible for instantiating the class
  - Creating the object
  - Usually initializes variables

Constructor

- Called using new operator:
  - `Box usefulBox = new Box();`
- There is a default (non-declared) constructor for every class
  - This is used if you do not write a constructor
  - Constructors with parameters will override this one, however
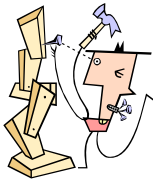
About Java Classes

- One "public" class per file
  - Public classes are available everywhere
- All code is contained in classes
  - File name is the public class name
    - Spelled exactly the same
    - Upper/lower case exactly the same
- Each public class is stored in its own source file
  - Has exactly same name as class
  - Source code uses .java extension
  - Compiled code uses a .class extension
- Used to create objects, run code, or serve as superclasses
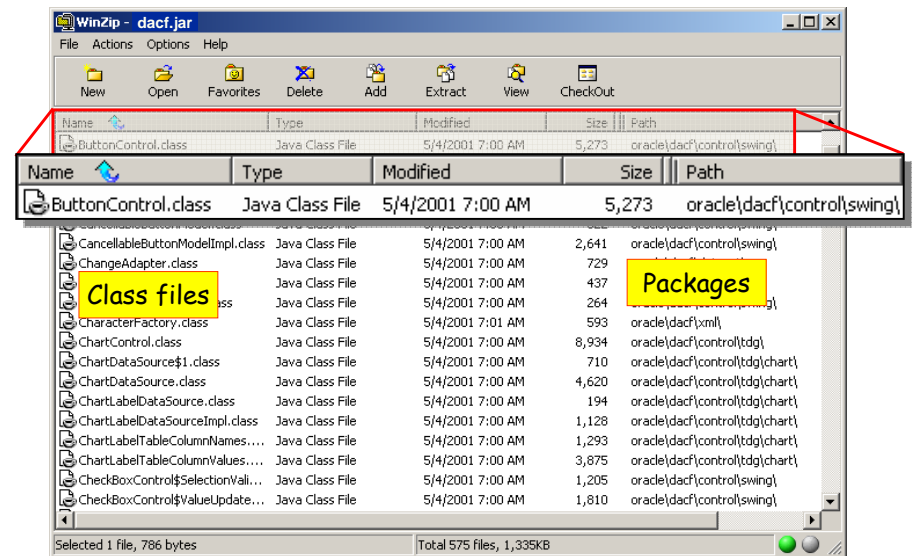
Using Java Classes

- To create an object, declare an instance
  - For example,
    `String empName = new String();`
  - This creates an object, empName
- Class files are collected into packages
  - Directories in the file system or in a zip file
- Java Archive (JAR) contains multiple class files
  - Can use .jar or .zip extension
  - "Libraries" made of one or more JARs

Sample Archive Contents

Name	Type	Modified	Size	Path
ButtonControl.class	Java Class File	5/4/2001 7:00 AM	5,273	oracle\dacf\control\swing\

Class files

Packages

CancellableButtonModelImpl.class	Java Class File	5/4/2001 7:00 AM	2,641	oracle\dacf\control\swing\
ChangeAdapter.class	Java Class File	5/4/2001 7:00 AM	729	
	Java Class File	5/4/2001 7:00 AM	437	
...ass	Java Class File	5/4/2001 7:00 AM	264	oracle\dacf\
CharacterFactory.class	Java Class File	5/4/2001 7:01 AM	593	oracle\dacf\xml\
ChartControl.class	Java Class File	5/4/2001 7:00 AM	8,934	oracle\dacf\control\tdg\
ChartDataSource$1.class	Java Class File	5/4/2001 7:00 AM	710	oracle\dacf\control\tdg\chart\
ChartDataSource.class	Java Class File	5/4/2001 7:00 AM	4,620	oracle\dacf\control\tdg\chart\
ChartLabelDataSource.class	Java Class File	5/4/2001 7:00 AM	194	oracle\dacf\control\tdg\chart\
ChartLabelDataSourceImpl.class	Java Class File	5/4/2001 7:00 AM	1,128	oracle\dacf\control\tdg\chart\
ChartLabelTableColumnNames....	Java Class File	5/4/2001 7:00 AM	1,293	oracle\dacf\control\tdg\chart\
ChartLabelTableColumnValues....	Java Class File	5/4/2001 7:00 AM	3,875	oracle\dacf\control\tdg\chart\
CheckBoxControl$SelectionVali...	Java Class File	5/4/2001 7:00 AM	1,205	oracle\dacf\control\swing\
CheckBoxControl$ValueUpdate...	Java Class File	5/4/2001 7:00 AM	1,810	oracle\dacf\control\swing\

Selected 1 file, 786 bytes   Total 575 files, 1,335KB

Naming Conventions

- Java is a case-sensitive language
  - Keywords are in lower case
    - **for, while, if, switch,** etc.
    - Case compliance is enforced for keywords

> **Note**: Java names can have any number of characters.

- There are conventions for other names
  - Normally, no underscores used
    - For example, EmpLastName not EMP_LAST_NAME
  - Package names are all lower case
  - Class names are mixed case ("camel case")
    - **EmployeeDataAccess**
  - Method and variable names are init-lower
    - **numberOfEmps, getCity(), setCity()**
  - Constants use all uppercase and underscores
    - MAX_LOAD, MIN_HEIGHT

Blocks and Comments

- Executable program blocks - { } symbols
  - Collection of declarations, specifiers, and methods
  - Code blocks can be nested
- Comments:
  - Single line
    ```
    // This is a single-line comment.
    int count;  // it can end a line
    ```
  - Multiline
    ```
    /* This is a multiline comment in
       Java, the same as in SQL. */
    /* It can be one line */
    ```
  - Javadoc
    ```
    /** This will be written to an
        HTML document. */
    ```

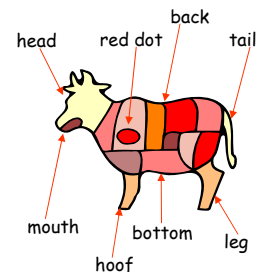# Simple Class Example

```
public class HiThere {
   public static void main (String[] args) {
      System.out.println("What's Happening?");
   }
}
```

- First line declares the class
  - Specifier **public** — available everywhere
    - Other specifiers: private, default, protected
  - { } represent the start and end of the code block
- Second line declares a method – the *method signature*
  - JVM looks for method **main()** when application starts
  - **void** declares a return type of nothing
  - Remainder used to pass parameters to **main()** method
- Third line calls external method to show message in console – command line window
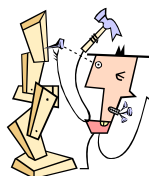
Anatomy of a Class

- Package that the class belongs to
- Import statements for libraries used
- Class declaration
- Variable declaration
- Methods and constructors
  - Constructor
    - Same name as class
    - Creates the object and initializes the data
  - **main()**
  - **set()** and **get()**
    - Called "accessors" or "getters and setters"
  - Application-specific methods

head red dot back tail mouth bottom hoof leg

Example Class

```
package shapes;

public class Rectangle {
  private int height;
  private int width;

  public Rectangle() {
    height = 1;
    width = 1;
  }

  public int getHeight() {
    return height;
  }

  public void setHeight(int newHeight) {
    height = newHeight;
  }

  public int getWidth() {
    return width;
  }

  public void setWidth(int newWidth) {
    width = newWidth;
  }
}
```

- Package statement
- Class declaration
- Constructor
- Variable declarations (attributes, fields)
- Code block symbol

Example Subclass

```
package shapes;
import java.util.*;

public class Box extends Rectangle {
  private int depth;
  private int height;

  public Box() {
    this.height = 4;
    super.setWidth(3);
    depth = 2;
  }

  public int getDepth() {
    return depth;
  }

  public void setDepth(int newDepth) {
    depth = newDepth;
  }

  public int boxVolume() {
    return height * getWidth() * getDepth();
  }
}
```

- Class imports
- Subclass keyword
- Variables and methods are called "members" of the class.
- set() and get() methods

Using Box

```
public class TestBox {
  public static void main(String[] args) {
    Box usefulBox = new Box();
    // getHeight() is from Rectangle
    // getDepth() is from Box
    // height in Box is not accessible
    System.out.println(
      "The height is " + usefulBox.getHeight());
    System.out.println(
      "The depth is " + usefulBox.getDepth());
    usefulBox.setWidth(5);
    System.out.println(
      "The volume is " + usefulBox.boxVolume());
  }
}
```

- main() method
- Object instantiation. Calls Box() which calls Rectangle()
- Call to method in external package

Output

```
The height is 1
The depth is 2
The volume is 40
```

Some Java Operators

Function	PL/SQL	Java
Concatenation	`||`	`+`
Modulus (remainder)	`MOD()`	`%`
Assignment	`:=`	`=`
Increment	`i := i + 1`	`i++`
Addition assignment	`i := i + 5`	`i += 5`
Equal to	`=`	`==`
Not equal to	`!=`	`!=`
Logical AND	`AND`	`&&`
Logical OR	`OR`	`||`
Ternary if-then-else	`DECODE()`	`? :`
Bitwise unary not	[nothing]	`~`
Percolate	`BREW()`	☺

Agenda

- Java Basics

- Object Orientation

- Anatomy 101

- Datatypes and Variables

- Control Statements
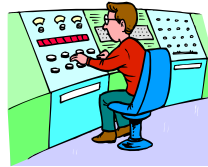
Variable Declarations

- You can declare multiple variables on one line
  ```
  int i, j, k;          ←———— declaration
  int i = 1;            ←———— initialization
  ```
- You can initialize at the same time
  ```
  int i = 2, j, k = 10;
  ```
- Variable and object declarations can take place anywhere
  - Java supports objects created on the fly
- Two datatype categories
  - Primitive: single value, no methods, no class
  - Reference: based on classes, technically a variable created from a class is an object

---

Primitive Types - Number

- Whole number
  - **byte**　　(-128 to 127)
  - **short**　(-32,768 to 32,767)
  - **int**　　(-2,147,483,648 to 2,147,483,647)
  - **long**　(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

  > 9.2 quintillion in North America
  > 9.2 trillion in Europe and the UK

- Decimal place
  - **float**　(3.4e-038 to 3.4e+038)
  - **double**　(1.7e-308 to 1.7e+308)
    - More precise than float, but takes twice the memory space (64 bytes)

Primitive Types – Character and Logical

- Character
  - **char** (integer of 16 bytes, 0 to 65,536)
  - Single character or symbol
  - Handles Unicode (an international character set)

  > The only character primitive datatype.

- Logical
  - **boolean (true** or **false)**
  - Two values only (no null logical value)
  - **true** is not a number like –1
  - No quotes around the symbol
  - For example:
    ```
    boolean isTrue = true;
    isTrue = (2 < 1);
    ```

char Examples

```
// decimal equivalent of letter 'a'
   char myFirstChar = 97;

// using a character
   char mySecondChar = 'a';

// octal equivalent of letter 'a'
   char myThirdChar = '\141';

// Unicode (Hex) value for 'a'
   char myFourthChar = '\u0061' ;
```

PL/SQL

```
v_string_char CHAR(66) := 'Data type CHAR is a fixed' ||
    ' length, multi-character string in PL/SQL';
```

Typing Based on a Class

- Use the new operator:

  ```
  String testString;          ← declaration
  testString = new String();  ← instantiation
  ```
  OR
  ```
  String somestring = new String();
  ```

  Use StringBuffer() if you will change the data

- Most any class can be used to create an object
  - Exceptions: abstract classes, classes with private constructors
  - Data and behavior of the class are available to the object
- *Wrapper classes* implement primitives
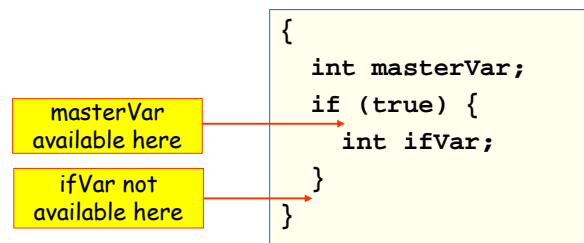 - These have methods (primitives do not)

Constants

- Use keyword **final**
  - Like **CONSTANT** in PL/SQL
  - Final variables must be initialized in same statement
  - For example,

    ```
    static final double PI = 3.141592;
    ```
- Can use **final** for methods
  - Method cannot be overridden in a subclass
- Can use **final** for classes
 - Final **classes** cannot be inherited

---

Variable Scope

- Variables last within the curly brackets or structure that encloses them

  ```
  for (int i = 0; i < 10; i++) {
  ```
  i available only during "for" loop

- Like PL/SQL nested blocks
- Curly brackets for if..else, loops count
  - Unlike PL/SQL

  ```
  {
      int masterVar;
      if (true) {
          int ifVar;
      }
  }
  ```
  masterVar available here

  ifVar not available here

Agenda

- Java Basics

- Object Orientation

- Anatomy 101

- Datatypes and Variables

- **Control Statements**

QUOVERA
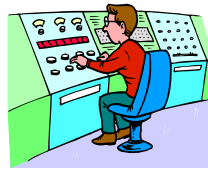
Standard Control Structures

- Sequence
  - Code executes in the order in which it is written
- Conditional branching
  - `if else, switch`
- Iteration
  - `while, for, do while`
- Jump statements
  - `break` - to exit a structure
  - `continue` – to start loop over
  - `return` – to go back to calling routine
  - No `goto`
- Exception handling
  - Enclose in `try {} catch {}` block
  - `throws` causes an explicit exception
  - Like PL/SQL BEGIN..EXCEPTION..END

QUOVERA
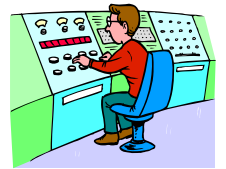
Conditional Example

```
class ShowQuarter {
  public static void main (String[] args) {
    int taxMonth = 10;
    String taxQuarter;

    if (taxMonth == 1 || taxMonth == 2 || taxMonth == 3) {
      taxQuarter = "1st Quarter";
      }
    else if (taxMonth >= 4 && taxMonth <= 6) {
      taxQuarter = "2nd Quarter";
      }
    else if (taxMonth >= 7 && taxMonth <= 9) {
      taxQuarter = "3rd Quarter";
      }
    else if (taxMonth >= 10 && taxMonth <= 12){
      taxQuarter = "4th Quarter";
      }
    else {
      taxQuarter = "Not Valid";
      }
    System.out.println("Your current Tax Quarter is: " +
      taxQuarter );
    }
}
```

comparison equals

Logical OR

Logical AND

Loop Examples

```
class DemoFor {
  public static void main (String[] args) {
    int i;
    for (i = 1; i <= 10; i++) {
      System.out.println("Loop count is " + i);
    }
  }
}
```

println() handles mixing of data types

increment operator

Could be: `for (int i = 1; i <= 10; i++)`

```
class DemoWhile {
  public static void main (String[] args) {
    int i = 1;

    while (i <= 10) {
      System.out.println(
        "Loop count is " + i);
      i++;
    }
  }
}
```

```
Loop count is 1
Loop count is 2
Loop count is 3
Loop count is 4
Loop count is 5
Loop count is 6
Loop count is 7
Loop count is 8
Loop count is 9
Loop count is 10
```

QUOVERA

Exception Handling Example

```java
public class TestException extends Object {
  public static void main(String[] args) {
    int numerator = 5, denominator = 0, ratio;
    try {
      ratio = numerator / denominator;
      System.out.println("The ratio is " + ratio);
    }
    catch (SQLException sqlexcept) {
      // display SQL error message
      sqlexcept.printStackTrace();
    }
    catch (Exception except) {
      // display generic error message
      except.printStackTrace();
    }
    finally {
      System.out.println("After finally.");
    }
    System.out.println("The end.");
  }
}
```

Will throw a divide-by-zero error.

Always run

Not run if an unhandled exception occurs

Some Gotchas

- Automatic rounding – use casting or correct datatype
  - E.g., the result of an int/int is an int:

  ```java
  int numA = 2, numB=3;
  System.out.println("result=" + numB/numA);
  ```
  result=1

- Be careful of order of precedence – use ()

  ```java
  System.out.println("result1=" + 4 + 5);
  System.out.println("result2=" + (4 + 5));
  ```
  result1=45
  result2=9

- Use {} around all if statement clauses
  - Only one statement executes after **if**

  ```java
  if (anInteger == 0)
    anotherInteger = 5;
    someInteger = 10;
  ```
  This always executes

Bibliography

- *Java 2, The Complete Reference, 5th Ed*
  - Herb Schildt, Osborne McGraw-Hill
- *Thinking in Java* (online or hard copy)
  - Bruce Eckels, www.mindview.net
- *Head First Java*
  - Kathy Sierra, Bert Bates, O'Reilly
- *Java Tutorial*
  - java.sun.com
- *Refactoring: Improving the Design of Existing Code*
  - Martin Fowler, Addison-Wesley

Quote

Real programmers
can write assembly code
in any language.

—Larry Wall

Summary

- Java has the basic language elements
- Java is a case-sensitive language
- All Java code is inside classes
- Classes are grouped into packages
- Variables can be typed from primitive data types or from classes
- Recognized naming conventions
- Other than syntax, the big difference between Java and PL/SQL is OO

- Please fill out the evals
- Books co-authored with Dr. Paul Dorsey, Avrom Roy-Faderman, & Duncan Mills
- Personal web site:
  `http://ourworld.compuserve.com/homepages/Peter_Koletzke`

Quovera

http://www.quovera.com

- Founded in 1995 as Millennia Vision Corp.
- Profitable for 7+ years without outside funding
- Consultants each have 10+ years industry experience
- Strong High-Tech industry background
- 200+ clients/300+ projects
- JDeveloper Partner
- More technical white papers and presentations on the web site

---

Quiz Time

1. Is **for** and **FOR** the same in Java?
2. How do you define a procedure in Java?
3. What is a constructor?
4. What's wrong with the following Java?
   string EmpName = 'SCOTT';
5. What does this mean?
   public static void main()

Quiz Time

6. What does the symbol "Test" represent in the following?
   ```
   public class Test
   public Test()
   public void test()
   int test;
   ```
7. What is the main container for Java code?
8. What is a subclass?
9. What is a method?

Quiz Answers

1. No. Java is case sensitive.
2. Java only has methods not procedures, but a method with a void return corresponds to a PL/SQL procedure.
3. It is a code unit used to create an object.
4. You need to use double quotes to delimit strings. Also, by convention, variables are initial lowercase. In addition, the standard string class is spelled "String".
5. The method, main, is available to all callers (public). It returns nothing (void) and you do not need an object to call it (static). In addition, if this signature had a String array for an argument, you could call main() from the command line by running the class.

Quiz Answers

6. These are, a class, a constructor, a method, and a variable, respectively.
7. The class is the direct container for Java code.
8. A subclass is a class declared as a child of another class using the keyword "extends".
9. A method is the main code unit in Java. Methods are contained in class files.